

Internalization of extensional equality

Andrew Polonsky

February 23, 2015

Abstract

We give a type system in which the universe of types is closed by reflection into it of the logical relation defined externally by induction on the structure of types. This contribution is placed in the context of the search for a natural, syntactic construction of the extensional equality type (Tait (1995), Altenkirch (1999), Coquand (2011), Licata and Harper (2012), Martin-Löf (2013)). The system is presented as an extension of λ^* , the terminal pure type system in which the universe of all types is a type. The universe inconsistency is then removed by the usual method of stratification into levels. We give a set-theoretic model for the stratified system. We conjecture that Strong Normalization holds as well.

1. Background: the problem of extensionality

In recent years, the problem of extensionality in type theory has received increasing attention. In part, this is due to type theory emerging as the language of choice for computer formalisation of mathematics. (Gonthier, Asperti, Avigad, Bertot, Cohen, Garillot, Roux, Mahboubi, O'Connor, Biha, Pasca, Rideau, Solovyev, Tassi and Théry (2013), The Univalent Foundations Program (n.d.).)

The fundamental notion of this language, that of a *type*, is a notion of collection which bases membership on the syntactic form of the objects. Accordingly, the notion of equality between objects of a given type is likewise based on their syntactic form: two expressions are judged as denoting equal objects if one can be transformed into another by a finite sequence of syntactic manipulations.

Since a general number-theoretic function can in principle be implemented in any number of ways, there will be different expressions defining the same function which cannot be transformed from one to another using syntactic manipulations only. For example, the function which maps a vector of numbers to a rearrangement of it listing the numbers in non-decreasing order, can be implemented using bubble-sort or quick-sort processes, and these cannot be transformed into one another by local simplifications.

Therefore, basing equality on syntactic form alone leads to the failure of *function extensionality*, the principle stating that two functions are equal if they are pointwise equal:

$$\forall x, y \ (x =_A y \Rightarrow f x =_B g y) \quad \Longrightarrow \quad f =_{A \rightarrow B} g \quad (\text{FE})$$

At the same time, this principle is deeply embedded into the language and culture of mathematics. After set theory became the standard language of communicating mathematical ideas, the intuitive notion of “function” came to be understood through its encoding into

set theory — as a set of ordered pairs. Since sets are extensional almost by definition, so must be functions in the set-theoretic foundations.

In the usual mathematics, the above “principle” is therefore a matter of linguistics.¹

But not so in type theory. Elementary mathematical arguments often become clumsy when translated to type theory, because the principle (FE) is not available.

In order to develop set-theoretic mathematics in type theory, it would be convenient to have a notion of equality that justified the above principle. Such a notion may then be called *extensional equality*. Unfortunately, the known constructions of this notion result in violation of key design principles of type theory.

2. Approaches to extensionality

The classical approach originally pursued by Martin-Löf (1984) is to extend the definitional (syntactic) equality between expressions, by allowing expressions to be declared syntactically equal whenever the statement of their equality is proved in the system’s logic. (That is, mathematical equality is reflected back into the syntax.)

This fixed the problem with (FE), but the cost was too great: syntactic equality, being now dependent on propositions, became undecidable, and so did type-checking. But type theory, following Curry–Howard paradigm, identifies proving propositions with inhabiting types. Thus undecidability of type-checking means it is not possible to decide whether a purported proof is indeed a proof, making type theory useless as a foundational system.²

A more recent idea is due to Voevodsky (2006), who discovered a single sentence in the language of type theory which, when assumed as an axiom, makes the intensional identity type behave like the extensional one. The axiom is a strong form of *universe extensionality* (Hofmann and Streicher (1996)) — stating that isomorphic types are equal — and has many deep consequences for type theory, including (FE).

However, assuming an axiom in type theory without specifying its combinatory behavior with respect to other symbols results in the loss of another crucial property of *canonicity*. This property guarantees that every type-theoretic construction is conceptually computable, in the sense that every definition can be computed by trivial simplification steps to a value.

A computational interpretation for Voevodsky’s axiom has been recently given by Coquand and collaborators (Bezem, Coquand and Huber (2014)). Their solution is *semantic*: all relevant computations are performed in a constructive model of homotopy types. The notion of extensional equality is given meaning only implicitly, via its interpretation in the homotopy model of type theory.

It would appear worthwhile to have a native, type-theoretic construction of extensional equality which did not assume the univalence axiom or the homotopy interpretation.

¹At the time of writing these words, the definition of the word “function” given by Google is that it is an “expression with one or more variables”. This corresponds to the type-theoretic notion of a function — a lambda term — and not the set-theoretic notion (a set of pairs having certain properties).

The middle-school definition of a function as a “black box” which *transforms* its input into output is likewise more faithfully captured by the lambda calculus standpoint.

² After all, the point of a formal system is not to abstractly talk about the set

$$\{A \mid A \text{ is true}\}$$

but rather to offer *convenient* tools whereby membership (of some elements) in this set can be established with finite effort.

3. Extensional equality by induction on type structure

Yet another approach is to define extensional equality by induction on type structure. Here one defines the equality relation *externally* to the type system, by relating certain elements of the (free) term model of the system.

This is actually the oldest approach to the problem, having led Gandy (1956) to derive the *logical relations principle*, a basic tool in metatheoretic studies of type systems.

In 1995, in a paper titled “Extensional Equality in the Classical Theory of Types”, William Tait assumed the denotation of the notion of extensional equality to be the canonical equivalence relation defined by the logical relations principle. While this idea is certainly familiar to many researchers, it appears to have missed a general announcement.

Let it thus be made explicit.

EXTENSIONALITY THESIS. *The extensional equality type is the canonical equivalence relation defined between elements of the term model of type theory by induction on type structure.*

The challenge in realizing Tait’s program is that the equality relation, being defined externally, is a priori valued in the types of the meta-level. In order to talk about equality *within* the system, we must reflect this relation from the meta-level back into the syntax. This step is somewhat delicate, and indeed has restricted much work along these lines to “truncated” systems, in which the equality type *cannot* be iterated to yield an infinite tower

$$p, q : a \simeq_A b, \quad p, q : p \simeq_{a \simeq_A b} q, \quad \pi, \xi : p \simeq_{p \simeq_{a \simeq_A b} q} q, \quad \dots$$

At the same time, we certainly do want to iterate the equality type, in order for it to generalize Martin-Löf intensional identity type Id_A .

Herein lies our main contribution. We describe a type system $\lambda \simeq$ in which the logical relation is reflected into the type structure via a new type constructor, the type $A \simeq B$ of *type equalities* (between types A, B). The (unique) elimination rule for this type associates to every $e : A \simeq B$ a heterogeneous *dependent equality* $\sim e : A \rightarrow B \rightarrow *$. The introduction rules for this type assert that every type constructor preserves type equality, including type equality itself. The computation rules capture the logical conditions associated to the corresponding type constructor.

We give a complete proof of the *preservation theorem* for $\lambda \simeq$, which states that every expression preserves the (reflected) logical relation. In particular, every closed term $a : A$ is related to itself, and the type $a \simeq_A a$ lives in the same universe as A .

For detailed development, from the simply typed lambda calculus to the system presented here, we refer the reader to our earlier report (Polonsky (2014)).

The system $\lambda \simeq$ gives a satisfactory definition of extensional equality for *closed* types. In order for equality to really behave like a “type constructor”, much work remains to be done.

In the future, we would like to internalize the preservation operator, so that extensionality of terms could be witnessed internally (cf. Bernardy and Moulin (2012)). We also want equality to satisfy the higher-dimensional analogues of symmetry and transitivity: the *Kan filling conditions*. An ultimate benchmark of success would be to validate all of the axioms for equality isolated by Coquand (2011, p.34).

From now on, we use the words “extensional equality” in the sense of the thesis above.

4. Related work

To place our paper in context, we review some recent developments in extensional equality.

- *Observational Equality Now!*, Altenkirch, McBride and Swierstra (2007)

The authors give a complete treatment of the 1-dimensional theory of equality (setoid level), including symmetry, transitivity, and the relevant computation rules. The constructions are performed in a metatheory having the uniqueness of identity proofs (UIP) principle.

- *Equality and dependent type theory*, Coquand (2011)

These slides, which had a great influence on our own investigations, contain early ideas for computing with univalence, taking the syntactic rather than semantic route.

- *Canonicity for 2-dimensional type theory*, Licata and Harper (2012)

As stated in the title, this theory is truncated at level 2. Nevertheless, it gives a complete computational treatment of the groupoid operations. The authors assume propositional reflection in the metatheory.

- *Computational interpretation of parametricity*, Bernardy and Moulin (2012)

Parametricity theory is intimately connected to extensional equality.

One key difference obtains in the treatment of universes. In the context of parametricity, the relation on the universe associates to each pair of types the type of binary relations between them. In our notation, this would appear as the rewrite rule

$$(\sim^*)AB \longrightarrow (A \rightarrow B \rightarrow *)$$

When defining extensional equality, we want the relation on the universe to be type equality. This may still allow interpretation by weaker notions of equality — such as isomorphism or homotopy equivalence — but should certainly prohibit general relations between types.

Instead, the relation on the universe in $(\lambda \simeq)$ associates to types A, B the type of equalities between A and B :

$$(\sim^*)AB \longrightarrow A \simeq B$$

The type $A \simeq B$ is thought of as the type of codes for relations with certain properties; those properties are validated by various notions of “equivalence of types”.

Another difference is that the preservation theorem in parametricity results is not iterable: even when carried out in a “reflective” PTS, the witnesses of parametricity are typed in a higher universe than the original terms.

In contrast, when we stratify $\lambda \simeq$, the “parametricity witnesses” will actually be typed in a *lower* universe than the given terms. (This choice will be forced upon us by semantic considerations.)

On the other hand, Bernardy and Moulin (2012) go much further in internalization, reflecting the preservation operator into the syntax as well. In our case, the preservation map is only a meta-level operation on pseudoterms.

- P. Martin-Löf, lectures given at CMU, Martin-Löf (2013)

This talk series gives a systematic treatment of the (1-dimensional) relation model.

- *Internalization of the groupoid model*, Sozeau and Tabareau (2014)

A complete formalization of the 2-dimensional theory in the Coq proof assistant.

As compared to the previous results, our contribution internalizes the external logical relation in a way that neither limits the resulting theory to a low dimension, nor requires any axioms in the metatheory.

In the next section we present the system $\lambda\simeq$. Section 6 gives the proof of Tait’s extensionality theorem for $\lambda\simeq$. We use this theorem to derive extensional equality for closed types in Section 7. Afterwards, we stratify the system to make it consistent, and give a natural set-theoretic model.

5. $\lambda\simeq$

In this section, we describe a type theory in which extensionality of terms is witnessed by terms in the same system. Denoted by $\lambda\simeq$, the system is an extension of $\lambda*$, the “naive” dependent type theory, by a new type, called *type equality*. The typing rules for this type ensure that extensionality of every term is witnessed within the system.³

The choice of $\lambda*$ as the base system is motivated by the fact that, although inconsistent, this system is by far and away the simplest formulation of dependent type theory. We found that postponing proper universe management until the rules for the new type are clearly set out simplifies the presentation considerably.

Afterwards, the standard recipe for turning an inconsistent type theory into a consistent one by stratifying the universes may be applied, and the proofs given earlier remain valid. Stratification of $\lambda\simeq$ will be given in Section 8.

The system admits a meta-level operation

$$(\cdot)^* : \mathit{Terms}(\lambda\simeq) \rightarrow \mathit{Terms}(\lambda\simeq)$$

which raises by one the dimension of a given term. Using this operation, we prove a new, fully internal form of the *extensionality theorem* from Tait (1995).

The dependent version of the theorem requires one to consider a certain relation on the universe of types, and for every pair of types related by it, a new “heterogeneous” relation between terms of these types. We shall now define these concepts.

Intuitive description

We set out by stipulating that there be a binary relation $\simeq : * \rightarrow * \rightarrow *$ on the universe of types. It is a new type constructor, and we call it *type equality*. For types $A, B : *$, the type of equalities between A and B is denoted $A \simeq B$.

Every equality $e : A \simeq B$ between A and B induces a binary relation $\sim_e : A \rightarrow B \rightarrow *$ between A and B . The $\sim(\cdot)$ -operator is the elimination rule for the type $A \simeq B$.

³ There is some reason to believe that $\lambda\simeq$ is a minimal dependent type theory with this property, since it is obtained from the “canonical” PTS $\lambda*$ by closing the type structure under reflection of the standard logical relation. (See (Polonsky (2014)) for additional commentary.)

For $a : A$ and $b : B$, we think of the type $\sim_e ab$ as representing equalities between a and b which are “lying over” $e : A \simeq B$. To articulate this intuition, we introduce the notation

$$a \sim_e b \quad := \quad \sim_e ab$$

We add term constructors which assert that every type constructor preserves type equality, including type equality itself. These terms are the constructors for the type $A \simeq B$.

For instance, the constructor corresponding to \simeq asserts that \simeq preserves type equality. The corresponding introduction rule becomes

$$\frac{A^* : A \simeq A' \quad B^* : B \simeq B'}{\simeq^* A^* B^* : (A \simeq B) \simeq (A' \simeq B')}$$

Finally, for every combination of an introduction rule and elimination rule, there must be a reduction rule specifying how the two interact. In λ_{\simeq} , there are four type constructors: Π , Σ , \simeq , and $*$. Thus, there are $4 \times 1 = 4$ reduction rules for \simeq .

The reduction rules capture the logical conditions corresponding to the four type constructors. They will insure that the extensional equality on every type is *definitionally* equal to a type expressible with basic constructors.⁴

For instance, extensional equality on the Π and Σ types is

$$\begin{aligned} f \simeq_{\Pi x:A.B(x)} f' &:= \Pi a:A \Pi a':A \Pi a^*:a \simeq_A a'. \quad fa \sim_{B(a^*)} f'a' \\ (a,b) \simeq_{\Sigma x:A.B(x)} (a',b') &:= \Sigma a^*:a \simeq_A a'. \quad b \sim_{B(a^*)} b' \end{aligned}$$

Leaving the full treatment of extensional equality to Section 7, suffice it to say that the reduction rules for type equality are wholly motivated by generalizing these “logical conditions” to the dependent case. Conversely, our definition of extensional equality will indeed arise as the specialization of the heterogeneous relation $\sim_e : A \rightarrow B \rightarrow *$ to the case when $A = B$ and when e is the “degenerate path” $r(A) : A \simeq A$.

We are ready to present the system λ_{\simeq} .

Formal description

Syntax :

$$\begin{aligned} A, B, s, t, e ::= & * \mid x \mid \Pi x:A.B \mid \Sigma x:A.B \mid A \simeq B \mid a \sim_e b \\ & \mid \lambda x:A.t \mid st \mid (s, t) \mid \pi_1 t \mid \pi_2 t \\ & \mid ** \mid \Pi^*[x, x', x^*] : A.B \mid \Sigma^*[x, x', x^*] : A.B \mid \simeq^* ee \end{aligned}$$

Typing (greyed out font demarcates implicit arguments):

$$\begin{aligned} & \frac{}{\Gamma \vdash * : *} \\ & \frac{\Gamma \vdash A : *}{\Gamma, x : A \vdash x : A} \\ & \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : *}{\Gamma, y : B \vdash M : A} \end{aligned}$$

⁴ This is also the reason why the type $a \sim_e b$ does not require axioms stating that it preserves equality: it is not a proper type constructor, but reduces to more basic types according to the structure of e .

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x : A. B : *}$$

$$\Gamma \vdash \Sigma x : A. B : *$$

$$\frac{\Gamma \vdash A : * \quad \Gamma \vdash B : *}{\Gamma \vdash A \simeq B : *}$$

$$\frac{\Gamma \vdash A : * \quad \Gamma \vdash B : * \quad \Gamma \vdash e : A \simeq B}{\Gamma \vdash \sim e : A \rightarrow B \rightarrow *}$$

$$\boxed{a \sim_e b \quad := \quad \sim e a b}$$

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A. b : \Pi x : A. B}$$

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma \vdash f : \Pi x : A. B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B[a/x]}$$

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : \Sigma x : A. B}$$

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma \vdash p : \Sigma x : A. B}{\Gamma \vdash \pi_1 p : A}$$

$$\Gamma \vdash \pi_2 p : B[\pi_1 p / x]$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : * \quad A = B}{\Gamma \vdash M : B}$$

$$\frac{}{\Gamma \vdash *^* : * \simeq *}$$

$$\frac{\begin{array}{l} \Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \\ \Gamma \vdash A' : * \quad \Gamma, x' : A' \vdash B' : * \\ \Gamma \vdash A^* : A \simeq A' \quad \Gamma, x : A, x' : A', x^* : x \sim_A x' \vdash B^* : B \simeq B' \end{array}}{\begin{array}{l} \Gamma \vdash \Pi^* [x, x', x^*] : A^*. B^* : \Pi x : A. B \simeq \Pi x' : A'. B' \\ \Gamma \vdash \Sigma^* [x, x', x^*] : A^*. B^* : \Sigma x : A. B \simeq \Sigma x' : A'. B' \end{array}}$$

$$\frac{\begin{array}{l} \Gamma \vdash A : * \quad \Gamma \vdash B : * \\ \Gamma \vdash A' : * \quad \Gamma \vdash B' : * \\ \Gamma \vdash A^* : A \simeq A' \quad \Gamma \vdash B^* : B \simeq B' \end{array}}{\Gamma \vdash \simeq^* A^* B^* : (A \simeq B) \simeq (A' \simeq B')}$$

Reduction :

$$\begin{aligned}
(\lambda x:A.s)t &\longrightarrow s[t/x] \\
\pi_i(s_1, s_2) &\longrightarrow s_i \\
A \sim_{**} B &\longrightarrow A \simeq B \\
f \sim_{\Pi^*}[x, x', x^*]:A^*.B^* f' &\longrightarrow \Pi x:A \Pi x':A' \Pi x^*:x \sim_{A^*} x'. f x \sim_{B^*} f' x' \\
p \sim_{\Sigma^*}[x, x', x^*]:A^*.B^* p' &\longrightarrow \Sigma a^*: \pi_1 p \sim_{A^*} \pi_1 p'. \pi_2 p \sim_{B^*} [\pi_1 p, \pi_1 p', a^*/x, x', x^*] \pi_2 p' \\
e \sim_{\simeq^*} A^* B^* e' &\longrightarrow \Pi a:A \Pi a':A' \Pi a^*:a \sim_{A^*} a' \\
&\quad \Pi b:B \Pi b':B' \Pi b^*:b \sim_{B^*} b'. (a \sim_e b) \simeq (a' \sim_{e'} b')
\end{aligned}$$

In what follows, we will often see a pattern where three operations of the same type appear in a row, like the triple-product sequences in the last reduction rule. To reduce clutter in such expressions, we introduce the following notations.

$$\begin{aligned}
\Pi \left(\begin{array}{c} x:A \\ y:B \\ z:C \end{array} \right) T &:= \Pi x:A \Pi y:B \Pi z:C.T \\
\lambda \left(\begin{array}{c} x:A \\ y:B \\ z:C \end{array} \right) t &:= \lambda x:A \lambda y:B \lambda z:C.t \\
M \left(\begin{array}{c} N_1 \\ N_2 \\ N_3 \end{array} \right) &:= M N_1 N_2 N_3 \\
M \left[\begin{array}{c} a/x \\ b/y \\ c/z \end{array} \right] &:= M[a/x][b/y][c/z] \\
\Pi^* \left[\begin{array}{c} x \\ y \\ z \end{array} \right] : A^*.B^* &:= \Pi^*[x, y, z] : A^*.B^* \\
\Sigma^* \left[\begin{array}{c} x \\ y \\ z \end{array} \right] : A^*.B^* &:= \Sigma^*[x, y, z] : A^*.B^*
\end{aligned}$$

With these conventions, the reduction rules for the type $A \simeq B$ may be rendered as

$$\begin{aligned}
A \sim_{**} B &\longrightarrow A \simeq B \\
f \sim_{\Pi^*}[x, x', x^*]:A^*.B^* f' &\longrightarrow \Pi \left(\begin{array}{c} a:A \\ a':A' \\ a^*:a \sim_{A^*} a' \end{array} \right) \sim_{B^*} \left[\begin{array}{c} a/x \\ a'/x' \\ a^*/x^* \end{array} \right] f x f' x' \\
p \sim_{\Sigma^*}[x, x', x^*]:A^*.B^* p' &\longrightarrow \sum_{a^*: \pi_1 p \sim_{A^*} \pi_1 p'} \sim_{B^*} \left[\begin{array}{c} \pi_1 p/x \\ \pi_1 p'/x' \\ a^*/x^* \end{array} \right] \pi_2 p \pi_2 p' \\
e \sim_{\simeq^*} A^* B^* e' &\longrightarrow \Pi \left(\begin{array}{c} a:A \\ a':A' \\ a^*:a \sim_{A^*} a' \end{array} \right) \Pi \left(\begin{array}{c} b:B \\ b':B' \\ b^*:b \sim_{B^*} b' \end{array} \right) (a \sim_e b) \simeq (a' \sim_{e'} b')
\end{aligned}$$

6. The $(\cdot)^*$ -operator

We now define the map $(\cdot)^* : \mathcal{T}erms(\lambda \simeq) \rightarrow \mathcal{T}erms(\lambda \simeq)$ which will satisfy

$$t = t(x_1, \dots, x_n) : A(\vec{x}) \implies t^* = t^* \left(\begin{array}{ccc} x_1 & \dots & x_n \\ x'_1 & \dots & x'_n \\ x^*_1 & \dots & x^*_n \end{array} \right) : t(\vec{x}) \sim_{A^*}(\vec{x}, \vec{x}', \vec{x}^*) t'(\vec{x}')$$

The intuition is that t^* gives the transport of t over a “formal path” in the context.

DEFINITION. Let $t \mapsto t'$ be the operation of apostrophizing every variable, bound or otherwise.

LEMMA.

- $(M[N/x])' = M'[N'/x']$
- $M = N \implies M' = N'$
- $\Gamma \vdash M : A \implies \Gamma' \vdash M' : A'$

Proof. Typography. □

DEFINITION. The operation $t \mapsto t^*$ is defined by induction on term structure.

In the equations that follow, the symbols A_* , B_* , a_* , etc. are free variables: the appearance of $*$ in a **subscript** is merely a suggestive choice of naming the variables.

$$\begin{aligned}
(*)^* &= *^* \\
(x)^* &= x^* \\
(\Pi x:A.B)^* &= \Pi^* \left[\begin{array}{c} x \\ x' \\ x^* \end{array} \right] : A^* . B^* \\
(\Sigma x:A.B)^* &= \Sigma^* \left[\begin{array}{c} x \\ x' \\ x^* \end{array} \right] : A^* . B^* \\
(A \simeq B)^* &= \simeq^* A^* B^* \\
(\sim e)^* &= e^* \\
(\lambda x:A. \lambda x':A'. \lambda x^*:x \sim_{A^*} x'. b^*)^* &= \lambda x:A. \lambda x':A'. \lambda x^*:x \sim_{A^*} x'. b^* \\
(fa)^* &= f^* a a' a^* \\
(a, b)^* &= (a^*, b^*) \\
(\pi_1 p)^* &= \pi_1 p^* \\
(\pi_2 p)^* &= \pi_2 p^* \\
(**)^* &= \lambda \left(\begin{array}{c} A : * \\ A' : * \\ A^* : A \simeq A' \end{array} \right) \lambda \left(\begin{array}{c} B : * \\ B' : * \\ B^* : B \simeq B' \end{array} \right) . \simeq^* A^* B^* \\
(\Pi^*[x, x_1, x_*] : A_* . B_*)^* &= \lambda \left(\begin{array}{c} f : \Pi x:A.B \\ f' : \Pi x':A'.B' \\ f^* : f \sim_{\Pi^* A^* B^*} f' \end{array} \right) \lambda \left(\begin{array}{c} f_1 : \Pi x_1:A_1.B_1 \\ f'_1 : \Pi x'_1:A'_1.B'_1 \\ f^*_1 : f_1 \sim_{\Pi^* A_1^* B_1^*} f'_1 \end{array} \right) . \\
&\quad \Pi^* \left[\begin{array}{c} a \\ a' \\ a^* \end{array} \right] : A^* \quad \Pi^* \left[\begin{array}{c} a_1 \\ a'_1 \\ a^*_1 \end{array} \right] : A_1^* \quad \Pi^* \left[\begin{array}{c} a_* \\ a'_* \\ a^*_* \end{array} \right] : A_*^* \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \left(\begin{array}{c} a_1 \\ a'_1 \\ a^*_1 \end{array} \right) . \\
&\quad B_*^* \left[\begin{array}{c} a/x \\ a'/x' \\ a^*/x^* \end{array} \right] \left[\begin{array}{c} a_1/x_1 \\ a'_1/x'_1 \\ a^*_1/x^*_1 \end{array} \right] \left[\begin{array}{c} a_*/x_* \\ a'_*/x'_* \\ a^*_/x^*_* \end{array} \right] \left(\begin{array}{c} fa \\ f'a' \\ f^*aa'a^* \end{array} \right) \left(\begin{array}{c} f_1a_1 \\ f'_1a'_1 \\ f^*_1a^*_1a^*_1 \end{array} \right) \\
(\Sigma^*[x, x_1, x_*] : A_* . B_*)^* &= \lambda \left(\begin{array}{c} p : \Sigma x:A.B \\ p' : \Sigma x':A'.B' \\ p^* : p \sim_{\Sigma^* A^* B^*} p' \end{array} \right) \lambda \left(\begin{array}{c} p_1 : \Sigma x_1:A_1.B_1 \\ p'_1 : \Sigma x'_1:A'_1.B'_1 \\ p^*_1 : p_1 \sim_{\Sigma^* A_1^* B_1^*} p'_1 \end{array} \right) . \\
&\quad \Sigma^* \left[\begin{array}{c} a_* : \pi_1 p \sim_{A_*} \pi_1 p_1 \\ a'_* : \pi_1 p' \sim_{A'_*} \pi_1 p'_1 \\ a^*_* : a_* \sim_{(A_*^* \pi_1 p \pi_1 p_1)^*} a^*_* \end{array} \right] : A_*^* \left(\begin{array}{c} \pi_1 p \\ \pi_1 p' \\ \pi_1 p^* \end{array} \right) \left(\begin{array}{c} \pi_1 p_1 \\ \pi_1 p'_1 \\ \pi_1 p^*_1 \end{array} \right) . \\
&\quad B_*^* \left[\begin{array}{c} \pi_1 p/x \\ \pi_1 p'/x' \\ \pi_1 p^*/x^* \end{array} \right] \left[\begin{array}{c} \pi_1 p_1/x_1 \\ \pi_1 p'_1/x'_1 \\ \pi_1 p^*_1/x^*_1 \end{array} \right] \left[\begin{array}{c} a_*/x_* \\ a'_*/x'_* \\ a^*_/x^*_* \end{array} \right] \left(\begin{array}{c} \pi_2 p \\ \pi_2 p' \\ \pi_2 p^* \end{array} \right) \left(\begin{array}{c} \pi_2 p_1 \\ \pi_2 p'_1 \\ \pi_2 p^*_1 \end{array} \right)
\end{aligned}$$

$$\begin{aligned}
(\simeq^* A_* B_*)^* &= \lambda \left(\begin{array}{l} e : A \simeq B \\ e' : A' \simeq B' \\ e^* : e \sim_{\simeq^* A_* B_*} e' \end{array} \right) \lambda \left(\begin{array}{l} e_1 : A_1 \simeq B_1 \\ e'_1 : A'_1 \simeq B'_1 \\ e_1^* : e_1 \sim_{\simeq^* A'_1 B'_1} e'_1 \end{array} \right). \\
&\quad \Pi^* \left[\begin{array}{c} a \\ a' \\ a^* \end{array} \right] : A^* \Pi^* \left[\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right] : A_1^* \Pi^* \left[\begin{array}{c} a_* \\ a'_* \\ a_*^* \end{array} \right] : A_*^* \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \left(\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right) \\
&\quad \Pi^* \left[\begin{array}{c} b \\ b' \\ b^* \end{array} \right] : B^* \Pi^* \left[\begin{array}{c} b_1 \\ b'_1 \\ b_1^* \end{array} \right] : B_1^* \Pi^* \left[\begin{array}{c} b_* \\ b'_* \\ b_*^* \end{array} \right] : B_*^* \left(\begin{array}{c} b \\ b' \\ b^* \end{array} \right) \left(\begin{array}{c} b_1 \\ b'_1 \\ b_1^* \end{array} \right). \\
&\quad \simeq^* \left(e^* \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \left(\begin{array}{c} b \\ b' \\ b^* \end{array} \right) \right) \left(e_1^* \left(\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right) \left(\begin{array}{c} b_1 \\ b'_1 \\ b_1^* \end{array} \right) \right)
\end{aligned}$$

LEMMA. $(M[N/x])^* = M^*[N/x, N'/x', N^*/x^*]$

Proof. By induction on the structure of M . □

LEMMA. $M = N \implies M^* = N^*$

Proof. By induction on the length of the reduction–expansion sequence in $M = N$, it suffices to show

$$M \rightarrow N \implies M^* \twoheadrightarrow N^* \quad (1)$$

First we argue that it is enough to consider contractions at the root of the term.

Indeed, suppose that $M = C[s]$, $N = C[t]$, and $s \rightarrow t$ by contraction at the root.

Let $C_0 = C[x_0]$, where x_0 is fresh.

Using Lemma 6, we write

$$\begin{aligned}
M^* &= C_0[s/x_0]^* = C_0^*[s/x_0, s'/x'_0, s^*/x_0^*] \\
N^* &= C_0[t/x_0]^* = C_0^*[t/x_0, t'/x'_0, t^*/x_0^*]
\end{aligned}$$

Since β -reduction is itself a congruence, it suffices to verify that the instances of each variable are reducible. That $s \rightarrow t$ and $s' \twoheadrightarrow t'$ is clear; that $s^* \twoheadrightarrow t^*$ remains to be proved.

There is thus no loss of generality in assuming that the redex is contracted at the root.

We now treat each reduction rule in order.

β Given $s = (\lambda x : A.M)N \rightarrow M[N/x] = t$, we are to show that

$$((\lambda x : A.M)N)^* \twoheadrightarrow M[N/x]^*$$

Indeed,

$$\begin{aligned}
((\lambda x : A.M)N)^* &= (\lambda x : A.M)^* N N' N^* \\
&= (\lambda x : A \lambda x' : A' \lambda x^* : x \sim_{A^*} x'. M^*) N N' N^* \\
&\twoheadrightarrow M^*[N/x, N'/x', N^*/x^*] \\
&= M[N/x]^*
\end{aligned}$$

by Lemma 6. Thus $s^* \twoheadrightarrow t^*$.

β_Σ We have

$$\begin{aligned}
(\pi_1(M, N))^* &= \pi_1(M, N)^* = \pi_1(M^*, N^*) \rightarrow M^* \\
(\pi_2(M, N))^* &= \pi_2(M, N)^* = \pi_2(M^*, N^*) \rightarrow N^*
\end{aligned}$$

\star^* Consider

$$\sim_{\star^*} \longrightarrow \lambda A : \star \lambda B : \star . A \simeq B$$

We have

$$\begin{aligned} (\sim_{\star^*})^* &= \lambda A : \star \lambda A' : \star \lambda A^* : A \sim_{\star^*} A' \\ &\quad \lambda B : \star \lambda B' : \star \lambda B^* : B \sim_{\star^*} B' . \simeq^* A^* B^* \\ &= (\lambda A : \star \lambda B : \star . A \simeq B)^* \end{aligned}$$

Π^* Consider

$$\begin{aligned} &\sim(\Pi^*[x, x_1, x_*] : A_* . B_*) \\ &\longrightarrow \lambda f : \Pi x : A . B \lambda f_1 : \Pi x_1 : A_1 . B_1 . \\ &\quad \Pi a : A \Pi a_1 : A_1 \Pi a_* : a \sim_{A_*} a_1 . f a \sim_{B_*[aa_1a_*/xx_1x_*]} f_1 a_1 \end{aligned}$$

Let T be the reduct on the right. We have

$$\begin{aligned} &(\sim \Pi^*[x, x_1, x_*] : A_* . B_*)^* \\ &= \lambda \left[\begin{array}{c} f : \Pi x : A . B \\ f' : \Pi x' : A' . B' \\ f^* : f \sim_{\Pi^* A^* B^*} f' \end{array} \right] \lambda \left[\begin{array}{c} f_1 : \Pi x_1 : A_1 . B_1 \\ f'_1 : \Pi x'_1 : A'_1 . B'_1 \\ f_1^* : f_1 \sim_{\Pi^* A_1^* B_1^*} f'_1 \end{array} \right] . \\ &\quad \Pi^* \left[\begin{array}{c} a \\ a' \\ a^* \end{array} \right] : A^* \Pi^* \left[\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right] : A_1^* \Pi^* \left[\begin{array}{c} a_* \\ a'_* \\ a_*^* \end{array} \right] : A_*^* \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \left(\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right) . \\ &\quad B_*^* \left[\begin{array}{c} a/x \\ a'/x' \\ a^*/x^* \end{array} \right] \left[\begin{array}{c} a_1/x_1 \\ a'_1/x'_1 \\ a_1^*/x_1^* \end{array} \right] \left[\begin{array}{c} a_*/x_* \\ a'_*/x'_* \\ a_*^*/x_*^* \end{array} \right] \left(\begin{array}{c} f a \\ f' a' \\ f^* a a' a^* \end{array} \right) \left(\begin{array}{c} f_1 a_1 \\ f'_1 a'_1 \\ f_1^* a_1 a'_1 a_1^* \end{array} \right) \end{aligned}$$

By inspection, this is exactly T^* . Let's check the innermost quantifier:

$$\begin{aligned} &(\Pi a_* : a \sim_{A_*} a_1 . f a \sim_{B_*[aa_1a_*/xx_1x_*]} f_1 a_1)^* \\ &= \Pi^* \left[\begin{array}{c} a_* : a \sim_{A_*} a_1 \\ a'_* : a' \sim_{A'_*} a'_1 \\ a_*^* : a_* \sim_{A_*^*} a_*^* \end{array} \right] : A_*^* \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \left(\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right) . (\sim B_*[aa_1a_*/xx_1x_*])(f a)(f_1 a_1)^* \end{aligned}$$

(Here we used that $(a \sim_{A_*} a_1)^* = A_*^* a a' a^* a_1 a'_1 a_1^*$.) Indeed,

$$\begin{aligned} &(\sim B_*[aa_1a_*/xx_1x_*])(f a)(f_1 a_1)^* \\ &= (B_*[a/x][a_1/x_1][a_*/x_*])^* \left(\begin{array}{c} f a \\ f' a' \\ f^* a a' a^* \end{array} \right) \left(\begin{array}{c} f_1 a_1 \\ f'_1 a'_1 \\ f_1^* a_1 a'_1 a_1^* \end{array} \right) \\ &= B_*^* \left[\begin{array}{c} a/x \\ a'/x' \\ a^*/x^* \end{array} \right] \left[\begin{array}{c} a_1/x_1 \\ a'_1/x'_1 \\ a_1^*/x_1^* \end{array} \right] \left[\begin{array}{c} a_*/x_* \\ a'_*/x'_* \\ a_*^*/x_*^* \end{array} \right] \left(\begin{array}{c} f a \\ f' a' \\ f^* a a' a^* \end{array} \right) \left(\begin{array}{c} f_1 a_1 \\ f'_1 a'_1 \\ f_1^* a_1 a'_1 a_1^* \end{array} \right) \end{aligned}$$

as required.

Other cases are treated similarly. □

DEFINITION. Let $\Gamma = (x_1 : A_1, \dots, x_n : A_n)$ be a context. Put

$$\Gamma^* := \left(\begin{array}{ccc} x_1 : A_1 & \dots & x_n : A_n \\ x'_1 : A'_1 & \dots & x'_n : A'_n \\ x_1^* : x_1 \sim_{A_1^*} x'_1 & & x_n^* : x_n \sim_{A_n^*} x'_n \end{array} \right)$$

THEOREM. $\Gamma \vdash M : A \implies \Gamma^* \vdash M^* : M \sim_{A^*} M'$

Proof. We proceed by induction on $\Gamma \vdash M : A$.

Axiom For the axiom rule $\overline{\vdash * : *}$, we have

$$\vdash *^* : * \simeq *$$

By conversion rule, $\vdash *^* : * \sim_{**} *$.

Variable Given the derivation

$$\frac{\Gamma \vdash A : *}{\Gamma, x : A \vdash x : A}$$

we have, by induction hypothesis, that

$$\Gamma^* \vdash A^* : A \sim_{**} A'$$

and hence $A^* : A \simeq A'$.

Clearly, $\Gamma^* \supseteq \Gamma' \vdash A' : *$.

Then $(\Gamma, x : A)^* = (\Gamma^*, x : A, x' : A', x^* : x \sim_{A^*} x')$ is a valid context, and

$$(\Gamma, x : A)^* \vdash x^* : x \sim_{A^*} x'$$

Weakening Given the derivation

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : *}{\Gamma, y : B \vdash M : A}$$

the induction hypothesis gives that

$$\begin{aligned} \Gamma^* \vdash M^* : M \sim_{A^*} M' \\ \Gamma^* \vdash B^* : B \simeq B' \end{aligned}$$

Then $(\Gamma, y : B)^* = (\Gamma^*, y : B, y' : B', y^* : y \sim_{B^*} y')$ is a valid context, and

$$(\Gamma, y : B)^* \vdash M^* : M \sim_{A^*} M'$$

(by applying weakening thrice).

Formation of Π, Σ Suppose we are given

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x : A. B : *}$$

Induction gives

$$\begin{aligned} \Gamma^* \vdash A^* : A \simeq A' \\ \Gamma^*, x : A, x' : A', x^* : x \sim_{A^*} x' \vdash B^* : B \simeq B' \end{aligned}$$

By lemmata, we also have apostrophized versions of these:

$$\begin{aligned} \Gamma' \vdash A' : * \\ \Gamma', x' : A' \vdash B' : * \end{aligned}$$

Together, the given data, the primed version, and the inductive version, provide the hypotheses necessary for the application of the Π^* -rule:

$$\begin{array}{c}
\Gamma \vdash A : * \qquad \qquad \qquad \Gamma, x : A \vdash B : * \\
\Gamma \vdash A' : * \qquad \qquad \qquad \Gamma, x' : A' \vdash B' : * \\
\Gamma \vdash A^* : A \simeq A' \qquad \Gamma, x : A, x' : A', x^* : x \sim_{A^*} x' \vdash B^* : B \simeq B' \\
\hline
\Pi^* [x, x', x^*] : A^*. B^* : \Pi x : A. B \simeq \Pi x' : A'. B'
\end{array}$$

Since $(\Pi x : A. B)^* = \Pi^* [x, x', x^*] : A^*. B^*$, the above judgement has the desired form. The case of Σ -formation is treated congruently.

\simeq -Formation This is like the previous case, but easier; given

$$\frac{\Gamma \vdash A : * \quad \Gamma \vdash B : *}{\Gamma \vdash A \simeq B : *}$$

we have that $\Gamma' \vdash A' : *, \Gamma' \vdash B' : *$, and also, by induction, that

$$\begin{array}{c}
\Gamma^* \vdash A^* : A \simeq A' \\
\Gamma^* \vdash B^* : B \simeq B'
\end{array}$$

These data allow us to apply the \simeq^* -rule, yielding

$$\Gamma^* \vdash \simeq^* A^* B^* : (A \simeq B) \simeq (A' \simeq B')$$

which type converts to $(A \simeq B) \sim_{**} (A \simeq B)'$, as required.

\sim -Formation Suppose we are given

$$\frac{\Gamma \vdash A : * \quad \Gamma \vdash B : * \quad \Gamma \vdash e : A \simeq B}{\Gamma \vdash \sim e : A \rightarrow B \rightarrow *}$$

Then we have $\Gamma' \vdash \sim e' : A' \rightarrow B' \rightarrow *$, and by induction

$$\Gamma^* \vdash e^* : e \sim_{(A \simeq B)^*} e'$$

We work in Γ^* . Reducing the type of e^* , we get

$$e^* : \prod \left(\begin{array}{c} a : A \\ a' : A' \\ a^* : a \sim_{A^*} a' \end{array} \right) \prod \left(\begin{array}{c} b : B \\ b' : B' \\ b^* : b \sim_{B^*} b' \end{array} \right) (a \sim_e b) \simeq (a' \sim_{e'} b')$$

When written in explicit form, this looks like

$$e^* : \prod \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \prod \left(\begin{array}{c} b \\ b' \\ b^* \end{array} \right) (\sim e a b) \simeq (\sim e' a' b')$$

By definition of $(\cdot)^*$,

$$(A \rightarrow B \rightarrow *)^* = \Pi^* [x, x', x^*] : A^* \Pi^* [y, y', y^*] : B^*. *^*$$

Thus, for any $E : A \rightarrow B \rightarrow *, E' : A' \rightarrow B' \rightarrow *$, we find that

$$E \sim_{(A \rightarrow B \rightarrow *)^*} E' = \prod \left(\begin{array}{c} x : A \\ x' : A' \\ x^* : x \sim_{A^*} x' \end{array} \right) \prod \left(\begin{array}{c} y : B \\ y' : B' \\ y^* : y \sim_{B^*} y' \end{array} \right) E x y \simeq E' x' y'$$

In particular, the type of e^* is exactly

$$(\sim e) \sim_{(A \rightarrow B \rightarrow *)^*} (\sim e')$$

Since $(\sim e)^* = e^*$, we conclude that

$$\Gamma^* \vdash (\sim e)^* : (\sim e) \sim_{(A \rightarrow B \rightarrow *)^*} (\sim e)'$$

Abstraction Given

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A. b : \Pi x : A. B}$$

We have, by IH, that

$$\begin{aligned} \Gamma^* \vdash A^* : A \simeq A' \\ \Gamma^*, x : A, x' : A', x^* : x \sim_{A^*} x' \vdash B^* : B \simeq B' \\ \Gamma^*, x : A, x' : A', x^* : x \sim_{A^*} x' \vdash b^* : b \sim_{B^*} b' \end{aligned}$$

Observe that our target type converts as

$$\begin{aligned} & (\lambda x : A. b) \sim_{(\Pi x : A. B)^*} (\lambda x' : A'. b') \\ &= \prod \left(\begin{array}{l} a : A \\ a' : A' \\ a^* : a \sim_{A^*} a' \end{array} \right) (\lambda x : A. b) a \sim_{B^*[a, a', a^*/x, x', x^*]} (\lambda x' : A'. b') a' \\ &= \prod \left(\begin{array}{l} a : A \\ a' : A' \\ a^* : a \sim_{A^*} a' \end{array} \right) b[a/x] \sim_{B^*[a, a', a^*/x, x', x^*]} b'[a'/x'] \\ &=_{\alpha} \prod \left(\begin{array}{l} x : A \\ x' : A' \\ x^* : x \sim_{A^*} x' \end{array} \right) b \sim_{B^*} b' \end{aligned}$$

The first two induction hypotheses give us that this is a well-formed type. The third, after three applications of the abstraction rule, gives

$$\Gamma^* \vdash (\lambda x : A. \lambda x' : A'. \lambda x^* : x \sim_{A^*} x'. b^*) : (\Pi x : A. \Pi x' : A'. \Pi x^* : x \sim_{A^*} x'. b \sim_{B^*} b')$$

The subject of this judgement is equal to $(\lambda x : A. b)^*$.

The type predicate converts to $(\lambda x : A. b) \sim_{(\Pi x : A. B)^*} (\lambda x' : A'. b')$.

Application Suppose we are given

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma \vdash f : \Pi x : A. B \quad \Gamma \vdash a : A}{\Gamma \vdash fa : B[a/x]}$$

The induction hypotheses are

$$\begin{aligned} \Gamma^* \vdash A^* : A \simeq A' \\ \Gamma^*, x : A, x' : A', x^* : x \sim_{A^*} x' \vdash B^* : B \simeq B' \\ \Gamma^* \vdash f^* : f \sim_{(\Pi x : A. B)^*} f' \\ \Gamma^* \vdash a^* : a \sim_{A^*} a' \end{aligned} \tag{2}$$

Working in Γ^* , we need to show that

$$(fa)^* : fa \sim_{B[a/x]^*} f'a'$$

Equivalently, we need to show that

$$f^*aa'a^* : fa \sim_{B[a,a',a^*/x,x',x^*]} f'a' \quad (3)$$

(where we used the substitution lemma to rewrite $B[a/x]^*$).

Applying the conversion rule to (2) gives

$$f^* := \prod \left(\begin{array}{c} x : A \\ x' : A' \\ x^* : x \sim_{A^*} x' \end{array} \right) fx \sim_{B^*} f'x'$$

Then, by a triple use of the application rule, we have

$$f^*aa'a^* : fa \sim_{B^*[a,a',a^*/x,x',x^*]} f'a'$$

which is typographically consistent with (3).

Pairing Let us be given

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : \Sigma x : A. B}$$

We work in Γ^* . By induction, we have

$$\begin{aligned} a^* &: a \sim_{A^*} a' \\ b^* &: b \sim_{B[a/x]^*} b' \end{aligned}$$

We may rewrite the latter as

$$b^* : b \sim_{B^*[a,a',a^*/x,x',x^*]} b' \quad (4)$$

Using these data, the following sequence of judgements may be verified:

$$\begin{aligned} &(\Gamma, x : A)^* \vdash B^* : B \simeq B' \\ &\Gamma^*, x^* : a \sim_{A^*} a' \vdash B^*[a, a', x^*/x, x', x^*] : B[a/x] \simeq B'[a'/x'] \\ &\Gamma^*, x^* : a \sim_{A^*} a' \vdash b \sim_{B^*[a, a', x^*/x, x', x^*]} b' : * \\ &\Gamma^* \vdash b \sim_{B^*[a, a', a^*/x, x', x^*]} b' : * \\ &\Gamma^* \vdash (a^*, b^*) : \Sigma a^* : a \sim_{A^*} a'. b \sim_{B^*[a, a', a^*/x, x', x^*]} b' \\ &\Gamma^* \vdash (a^*, b^*) : (a, b) \sim_{\Sigma^*[x, x', x^*] : A^*. B^*} (a', b') \\ &\Gamma^* \vdash (a, b)^* : (a, b) \sim_{(\Sigma x : AB)^*} (a', b') \end{aligned}$$

Projections Next, we consider the inference rules

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma \vdash p : \Sigma x : A. B}{\Gamma \vdash \pi_1 p : A \quad \Gamma \vdash \pi_2 p : B[\pi_1 p/x]}$$

We have

$$\begin{aligned}
\Gamma^* \vdash p^* &: p \sim_{(\Sigma x:A.B)^*} p' \\
\Gamma^* \vdash p^* &: p \sim_{\Sigma[x,x',x^*]:A^*.B^*} p' \\
\Gamma^* \vdash p^* &: \Sigma a^* : \pi_1 p \sim_{A^*} \pi_1 p'. \pi_2 p \sim_{B^*[\pi_1 p, \pi_1 p', a^*/x, x', x^*]} \pi_2 p' \\
\Gamma^* \vdash \pi_1 p^* &: \pi_1 p \sim_{A^*} \pi_1 p' \\
\Gamma^* \vdash (\pi_1 p)^* &: (\pi_1 p) \sim_{A^*} (\pi_1 p)' \tag{p1}
\end{aligned}$$

$$\begin{aligned}
\Gamma^* \vdash \pi_2 p^* &: \pi_2 p \sim_{B^*[\pi_1 p, \pi_1 p', \pi_1 p^*/x, x', x^*]} \pi_2 p' \\
\Gamma^* \vdash (\pi_2 p)^* &: (\pi_2 p) \sim_{B^*[\pi_1 p, (\pi_1 p)', (\pi_1 p)^*/x, x', x^*]} (\pi_2 p)' \\
\Gamma^* \vdash (\pi_2 p)^* &: (\pi_2 p) \sim_{B[\pi_1 p/x]^*} (\pi_2 p)' \tag{p2}
\end{aligned}$$

The judgements (p1) and (p2) are of the required form.

Conversion Next, suppose we are given the inference

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : * \quad A = B}{\Gamma \vdash M : B}$$

We are to show that $\Gamma^* \vdash M^* : M \sim_{B^*} M'$.

From the given data, we know that

$$\begin{aligned}
M &: B \\
M' &: B' \\
B^* &: B \simeq B'
\end{aligned}$$

Thus

$$M \sim_{B^*} M' : * \tag{5}$$

By Lemma 1, we have

$$M \sim_{A^*} M' = M \sim_{B^*} M' \tag{6}$$

By IH, we also have $\Gamma^* \vdash M^* : M \sim_{A^*} M'$. Using (5) and (6), we may apply the conversion rule to this judgment to obtain

$$\Gamma^* \vdash M^* : M \sim_{B^*} M'$$

***-Congruence** Let us be given

$$\frac{}{** : * \simeq *}$$

We are asked to show that

$$\begin{aligned}
(*)^* &: ** \sim_{(* \simeq *)^*} ** \\
(*)^* &: ** \sim_{\simeq^{**} **} ** \\
(*)^* &: \prod \left(\begin{array}{c} A : * \\ A' : * \\ A^* : A \sim_{**} A' \end{array} \right) \prod \left(\begin{array}{c} B : * \\ B' : * \\ B^* : B \sim_{**} B' \end{array} \right). (A \sim_{**} B) \simeq (A' \sim_{**} B')
\end{aligned}$$

Unfolding the definition of $(*)^*$, we have

$$(*)^* = \lambda \left(\begin{array}{c} A : * \\ A' : * \\ A^* : A \simeq A' \end{array} \right) \lambda \left(\begin{array}{c} B : * \\ B' : * \\ B^* : B \simeq B' \end{array} \right). \simeq^* A^* B^*$$

By inspection, this term has the desired type.

Π -congruence Let us be given

$$\frac{\begin{array}{l} \Gamma \vdash A : * \\ \Gamma \vdash A_1 : * \\ \Gamma \vdash A_* : A \simeq A_1 \end{array} \quad \begin{array}{l} \Gamma, x : A \vdash B : * \\ \Gamma, x_1 : A_1 \vdash B_1 : * \\ \Gamma, x : A, x_1 : A_1, x_* : x \sim_{A_*} x_1 \vdash B_* : B \simeq B_1 \end{array}}{\Pi^*[x, x_1, x_*] : A_* . B_* : \Pi x : A . B \simeq \Pi x_1 : A_1 . B_1}$$

Recall that

$$\begin{aligned} & (\Pi^*[x, x_1, x_*] : A_* . B_*)^* \\ &= \lambda \left(\begin{array}{c} f : \Pi x : A . B \\ f' : \Pi x' : A' . B' \\ f^* : f \sim_{\Pi^* A^* B^*} f' \end{array} \right) \lambda \left(\begin{array}{c} f_1 : \Pi x_1 : A_1 . B_1 \\ f'_1 : \Pi x'_1 : A'_1 . B'_1 \\ f_1^* : f_1 \sim_{\Pi^* A_1^* B_1^*} f'_1 \end{array} \right). \\ & \Pi^* \left[\begin{array}{c} a \\ a' \\ a^* \end{array} \right] : A^* \quad \Pi^* \left[\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right] : A_1^* \quad \Pi^* \left[\begin{array}{c} a_* \\ a'_* \\ a_*^* \end{array} \right] : A_*^* \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \left(\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right). \\ & B_*^* \left[\begin{array}{c} a/x \\ a'/x' \\ a^*/x^* \end{array} \right] \left[\begin{array}{c} a_1/x_1 \\ a'_1/x'_1 \\ a_1^*/x_1^* \end{array} \right] \left[\begin{array}{c} a_*/x_* \\ a'_*/x'_* \\ a_*^*/x_*^* \end{array} \right] \left(\begin{array}{c} fa \\ f'a' \\ f^*aa'a^* \end{array} \right) \left(\begin{array}{c} f_1a_1 \\ f'_1a'_1 \\ f_1^*a_1^*a_1^* \end{array} \right) \end{aligned} \tag{7}$$

We are to show that this term has type

$$\Pi^* A_* B_* \sim_{\simeq^* (\Pi x : A . B)^* (\Pi x_1 : A_1 . B_1)^*} \Pi^* A'_* B'_* \tag{8}$$

First, note that the type of the equivalence appearing in the index of the dependent relation in (8) is

$$(\Pi x : A . B \simeq \Pi x_1 : A_1 . B_1) \simeq (\Pi x : A' . B' \simeq \Pi x_1 : A'_1 . B'_1)$$

(This is by the induction hypothesis on $\Pi x : A . B$ and $\Pi x_1 : A_1 . B_1$. Although “ $\Pi x : A . B : *$ ” does not appear among the premises of this rule, the required statement can be obtained by inlining the proof of the Π -formation case. The hypotheses there are provided by the induction hypotheses on the premises given here.)

We begin by looking closer at the relation associated to this equivalence. By \sim -reduction, we have

$$\begin{aligned} & \sim(\simeq^* (\Pi x : A . B)^* (\Pi x_1 : A_1 . B_1)^*) \\ &= \lambda e : (\Pi x : A . B \simeq \Pi x_1 : A_1 . B_1) \lambda e' : (\Pi x' : A' . B' \simeq \Pi x'_1 : A'_1 . B'_1). \\ & \quad \Pi \left(\begin{array}{c} f : \Pi x : A . B \\ f' : \Pi x' : A' . B' \\ f^* : f \sim_{(\Pi x : A . B)^*} f' \end{array} \right) \Pi \left(\begin{array}{c} f_1 : \Pi x_1 : A_1 . B_1 \\ f'_1 : \Pi x'_1 : A'_1 . B'_1 \\ f_1^* : f_1 \sim_{(\Pi x_1 : A_1 . B_1)^*} f'_1 \end{array} \right) \\ & \quad (f \sim_e f_1) \simeq (f' \sim_{e'} f'_1) \end{aligned}$$

When this term is applied to $\Pi^* A_* B_*$ and $\Pi^* A'_* B'_*$, so as to become (8), we see immediately that λ -abstractions appearing at the root of (7) match correctly the Π -types above.

We are thus left to verify that the matrix of these abstractions — the triple- Π^* subexpression of (7) — has type

$$\begin{aligned} (f \sim_{\Pi^* A_* B_*} f_1) &\simeq (f' \sim_{\Pi^* A'_* B'_*} f'_1) = \\ &\Pi a : A \Pi a_1 : A_1 \Pi a_* : a \sim_{A_*} a_1 \cdot (f a \sim_{B_*[a, a_1, a_*/x, x_1, x_*]} f_1 a_1) \\ &\simeq \Pi a' : A' \Pi a'_1 : A'_1 \Pi a'_* : a' \sim_{A'_*} a'_1 \cdot (f' a' \sim_{B'_*[a', a'_1, a'_*/x', x'_1, x'_*]} f'_1 a'_1) \end{aligned} \quad (9)$$

Looking at (7) again, we see that the equivalences specified in the triple- Π^* expression correctly match the domains of the products above.⁵ That the third equivalence

$$A_*^* \left(\begin{array}{c} a \\ a' \\ a_* \end{array} \right) \left(\begin{array}{c} a_1 \\ a'_1 \\ a_1 \end{array} \right) : (a \sim_{A_*} a_1) \simeq (a' \sim_{A'_*} a'_1)$$

has the right type uses induction hypothesis on A_* .

By induction hypothesis on B_* , we have that

$$(\Gamma, x : A, x_1 : A_1, x_* : x \sim_{A_*} x_1)^* \vdash B_*^* : B_* \sim_{\simeq^* B^* B_1^*} B'_*$$

or, using alternative notation,

$$(\Gamma, x : A, x_1 : A_1, x_* : x \sim_{A_*} x_1)^* \vdash B_*^* : (\simeq^* B^* B_1^*) \sim B_* B'_*$$

Plugging in the terms in our context, we get

$$\begin{aligned} \Gamma^* \vdash B_*^* &\left[\begin{array}{c} a/x \\ a'/x' \\ a^*/x^* \end{array} \right] \left[\begin{array}{c} a_1/x_1 \\ a'_1/x'_1 \\ a_1/x_1 \end{array} \right] \left[\begin{array}{c} a_*/x_* \\ a'_*/x'_* \\ a_*/x_* \end{array} \right] \\ &: \left(\simeq^* B^* \left[\begin{array}{c} a/x \\ a'/x' \\ a^*/x^* \end{array} \right] B_1^* \left[\begin{array}{c} a_1/x_1 \\ a'_1/x'_1 \\ a_1/x_1 \end{array} \right] \right) \sim B_*[\bar{a}/\bar{x}] B'_*[\bar{a}'/\bar{x}'] \\ &= \Pi \left(\begin{array}{c} y : B[a] \\ y' : B'[a'] \\ y^* : y \sim_{B^*[a, a', a^*]} y' \end{array} \right) \Pi \left(\begin{array}{c} y_1 : B_1[a_1] \\ y'_1 : B'_1[a'_1] \\ y_1^* : y_1 \sim_{B_1^*[a_1, a'_1, a_1^*]} y'_1 \end{array} \right) \cdot \\ &\quad (y \sim_{B_*[\bar{a}]} y_1) \simeq (y' \sim_{B'_*[\bar{a}']} y'_1) \end{aligned} \quad (10)$$

Next, we recall that

$$\begin{aligned} f^* &: \Pi \left(\begin{array}{c} x : A \\ x' : A' \\ x^* : x \sim_{A^*} x' \end{array} \right) f x \sim_{B^*} f' x' \\ f_1^* &: \Pi \left(\begin{array}{c} x_1 : A_1 \\ x'_1 : A'_1 \\ x_1^* : x_1 \sim_{A_1^*} x'_1 \end{array} \right) f_1 x_1 \sim_{B_1^*} f'_1 x'_1 \end{aligned}$$

⁵ We recall that in order to construct an equivalence between two Π -types, one needs to construct an equivalence between their domains of quantification, and, for every dependent line between these domains (ie, a pair of terms related by the equivalence), an equivalence between the corresponding fibers of the dependent type (quantification matrices).

This possibility is precisely the content of the Π^* -constructor, which gives an equivalence between two Π -terms from an equivalence between their domains and a map transporting paths between domains to equivalences of fibers.

It follows that

$$\begin{aligned} f^* a a' a^* : B^* \left[\begin{array}{c} a/x \\ a'/x' \\ a^*/x^* \end{array} \right] &\sim (fa)(f'a') \\ f_1^* a_1 a'_1 a_1^* : B_1^* \left[\begin{array}{c} a_1/x_1 \\ a'_1/x'_1 \\ a_1^*/x_1^* \end{array} \right] &\sim (f_1 a_1)(f'_1 a'_1) \end{aligned}$$

Putting this together with (10), we get

$$\begin{aligned} B_*^* \left[\begin{array}{c} a \\ a' \\ a^* \end{array} \right] \left[\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right] \left[\begin{array}{c} a_* \\ a_*' \\ a_*^* \end{array} \right] &\left(\begin{array}{c} fa \\ f'a' \\ f^* a a' a^* \end{array} \right) \left(\begin{array}{c} f_1 a_1 \\ f'_1 a'_1 \\ f_1^* a_1 a'_1 a_1^* \end{array} \right) \\ &: (fa \sim_{B_*[\bar{a}]} f_1 a_1) \simeq (f'a' \sim_{B_*'[\bar{a}']} f'_1 a'_1) \end{aligned}$$

This matches the expression in (9), concluding this case.

Σ -congruence Let us be given

$$\begin{array}{c} \Gamma \vdash A : * \qquad \qquad \qquad \Gamma, x : A \vdash B : * \\ \Gamma \vdash A_1 : * \qquad \qquad \qquad \Gamma, x_1 : A_1 \vdash B_1 : * \\ \Gamma \vdash A_* : A \simeq A_1 \quad \Gamma, x : A, x_1 : A_1, x_* : x \sim_{A_*} x_1 \vdash B_* : B \simeq B_1 \\ \hline \Sigma^* [x, x_1, x_*] : A_* . B_* : \Sigma x : A . B \simeq \Sigma x_1 : A_1 . B_1 \end{array}$$

We are to check that $(\Sigma^* [x, x_1, x_*] : A_* . B_*)^*$ has type

$$\begin{aligned} &(\Sigma^* [x, x_1, x_*] : A_* . B_*) \sim_{(\Sigma x : A . B \simeq \Sigma x_1 : A_1 . B_1)^*} (\Sigma^* [x, x_1, x_*] : A_* . B_*)' \\ &= (\Sigma^* [x, x_1, x_*] : A_* . B_*) \sim_{\simeq^*} (\Sigma x : A . B)^* (\Sigma x_1 : A_1 . B_1)^* (\Sigma^* [x', x'_1, x'_*] : A'_* . B'_*) \\ &= \prod \left(\begin{array}{c} p : \Sigma x : A . B \\ p' : \Sigma x' : A' . B' \\ p^* : p \sim_{(\Sigma x : A . B)^*} p' \end{array} \right) \prod \left(\begin{array}{c} p_1 : \Sigma x_1 : A_1 . B_1 \\ p'_1 : \Sigma x'_1 : A'_1 . B'_1 \\ p_1^* : p_1 \sim_{(\Sigma x_1 : A_1 . B_1)^*} p'_1 \end{array} \right). \\ &\quad (p \sim_{\Sigma^* [x, x_1, x_*] : A_* . B_*} p_1) \simeq (p' \sim_{\Sigma^* [x', x'_1, x'_*] : A'_* . B'_*} p'_1) \end{aligned}$$

Laying down this type on top of

$$\begin{aligned} &(\Sigma^* [x, x_1, x_*] : A_* . B_*)^* \\ &= \lambda \left(\begin{array}{c} p : \Sigma x : A . B \\ p' : \Sigma x' : A' . B' \\ p^* : p \sim_{\Sigma^* A_* B_*} p' \end{array} \right) \lambda \left(\begin{array}{c} p_1 : \Sigma x_1 : A_1 . B_1 \\ p'_1 : \Sigma x'_1 : A'_1 . B'_1 \\ p_1^* : p_1 \sim_{\Sigma^* A'_1 B'_1} p'_1 \end{array} \right). \\ &\quad \Sigma^* \left[\begin{array}{c} a_* : \pi_1 p \sim_{A_*} \pi_1 p_1 \\ a'_* : \pi_1 p' \sim_{A'_*} \pi_1 p'_1 \\ a_*^* : a_* \sim_{(A_* \sim_{\Sigma^* A_* B_*} A'_*)^*} a'_* \end{array} \right] : A_*^* \left(\begin{array}{c} \pi_1 p \\ \pi_1 p' \\ \pi_1 p^* \end{array} \right) \left(\begin{array}{c} \pi_1 p_1 \\ \pi_1 p'_1 \\ \pi_1 p_1^* \end{array} \right). \\ &\quad B_*^* \left[\begin{array}{c} \pi_1 p/x \\ \pi_1 p'/x' \\ \pi_1 p^*/x^* \end{array} \right] \left[\begin{array}{c} \pi_1 p_1/x_1 \\ \pi_1 p'_1/x'_1 \\ \pi_1 p_1^*/x_1^* \end{array} \right] \left[\begin{array}{c} a_*/x_* \\ a'_*/x'_* \\ a_*^*/x_*^* \end{array} \right] \left(\begin{array}{c} \pi_2 p \\ \pi_2 p' \\ \pi_2 p^* \end{array} \right) \left(\begin{array}{c} \pi_2 p_1 \\ \pi_2 p'_1 \\ \pi_2 p_1^* \end{array} \right) \end{aligned}$$

one can discern that the Π - and λ -binders have similar domains.

The terms will forever be united in a valid typing judgment if

$$\begin{aligned} &\Sigma^* \left[\begin{array}{c} a_* : \pi_1 p \sim_{A_*} \pi_1 p_1 \\ a'_* : \pi_1 p' \sim_{A'_*} \pi_1 p'_1 \\ a_*^* : a_* \sim_{(A_* \sim_{\Sigma^* A_* B_*} A'_*)^*} a'_* \end{array} \right] : A_*^* \left(\begin{array}{c} \pi_1 p \\ \pi_1 p' \\ \pi_1 p^* \end{array} \right) \left(\begin{array}{c} \pi_1 p_1 \\ \pi_1 p'_1 \\ \pi_1 p_1^* \end{array} \right). \\ &\quad B_*^* \left[\begin{array}{c} \pi_1 p/x \\ \pi_1 p'/x' \\ \pi_1 p^*/x^* \end{array} \right] \left[\begin{array}{c} \pi_1 p_1/x_1 \\ \pi_1 p'_1/x'_1 \\ \pi_1 p_1^*/x_1^* \end{array} \right] \left[\begin{array}{c} a_*/x_* \\ a'_*/x'_* \\ a_*^*/x_*^* \end{array} \right] \left(\begin{array}{c} \pi_2 p \\ \pi_2 p' \\ \pi_2 p^* \end{array} \right) \left(\begin{array}{c} \pi_2 p_1 \\ \pi_2 p'_1 \\ \pi_2 p_1^* \end{array} \right) \end{aligned}$$

has type

$$\begin{aligned}
& (p \sim_{\Sigma^*} [x, x_1, x_*] : A_* . B_* p_1) \simeq (p' \sim_{\Sigma^*} [x', x'_1, x'_*] : A'_* . B'_* p'_1) \\
& = \left(\sum^* \begin{bmatrix} x \\ x_1 \\ x_* \end{bmatrix} : A_* . B_* \right) \tilde{p} p_1 \simeq \left(\sum^* \begin{bmatrix} x' \\ x'_1 \\ x'_* \end{bmatrix} : A'_* . B'_* \right) \tilde{p}' p'_1 \\
& = \sum_{a_* : \pi_1 p \sim_{A_*} \pi_1 p_1} B_* \begin{bmatrix} \pi_1 p / x \\ \pi_1 p_1 / x_1 \\ a_* / x_* \end{bmatrix} \tilde{\pi}_2 p \pi_2 p_1 \simeq \sum_{a'_* : \pi_1 p' \sim_{A'_*} \pi_1 p'_1} B'_* \begin{bmatrix} \pi_1 p' / x' \\ \pi_1 p'_1 / x'_1 \\ a'_* / x'_* \end{bmatrix} \tilde{\pi}_2 p' \pi_2 p'_1
\end{aligned}$$

This is indeed the case, for by induction it so happens that

$$A_*^* \left(\begin{bmatrix} \pi_1 p \\ \pi_1 p' \\ \pi_1 p_* \end{bmatrix} \right) \left(\begin{bmatrix} \pi_1 p_1 \\ \pi_1 p'_1 \\ \pi_1 p_1 \end{bmatrix} \right) : (A_* \pi_1 p \pi_1 p_1) \simeq (A'_* \pi_1 p' \pi_1 p'_1)$$

$$\text{and, for } a_* : A_*^* \left(\begin{bmatrix} \pi_1 p \\ \pi_1 p' \\ \pi_1 p_* \end{bmatrix} \right) \tilde{a}_* a'_*,$$

$$\begin{aligned}
& B_*^* \begin{bmatrix} \pi_1 p / x \\ \pi_1 p' / x' \\ \pi_1 p_* / x_* \end{bmatrix} \left[\begin{bmatrix} \pi_1 p_1 / x_1 \\ \pi_1 p'_1 / x'_1 \\ \pi_1 p_1 / x_1 \end{bmatrix} \right] \left[\begin{bmatrix} a_* / x_* \\ a'_* / x'_* \\ a_* / x_* \end{bmatrix} \right] \left(\begin{bmatrix} \pi_2 p \\ \pi_2 p' \\ \pi_2 p_* \end{bmatrix} \right) \left(\begin{bmatrix} \pi_2 p_1 \\ \pi_2 p'_1 \\ \pi_2 p_1 \end{bmatrix} \right) \\
& : \sim B_* \begin{bmatrix} \pi_1 p / x \\ \pi_1 p_1 / x_1 \\ a_* / x_* \end{bmatrix} \tilde{\pi}_2 p \pi_2 p_1 \simeq \sim B'_* \begin{bmatrix} \pi_1 p' / x' \\ \pi_1 p'_1 / x'_1 \\ a'_* / x'_* \end{bmatrix} \tilde{\pi}_2 p' \pi_2 p'_1
\end{aligned}$$

This completes the case of Σ -congruence.

\simeq^* -congruence The last case left standing is the \simeq^* -constructor:

$$\frac{\begin{array}{cc} \Gamma \vdash A : * & \Gamma \vdash B : * \\ \Gamma \vdash A_1 : * & \Gamma \vdash B_1 : * \\ \Gamma \vdash A_* : A \simeq A_1 & \Gamma \vdash B_* : B \simeq B_1 \end{array}}{\simeq^* A_* B_* : (A \simeq B) \simeq (A_1 \simeq B_1)}$$

To get it down, we just need to force

$$\begin{aligned}
(\simeq^* A_* B_*)^* & = \lambda \left(\begin{array}{c} e : A \simeq B \\ e' : A' \simeq B' \\ e_* : e \sim_{\simeq^* A_* B_*} e' \end{array} \right) \lambda \left(\begin{array}{c} e_1 : A_1 \simeq B_1 \\ e'_1 : A'_1 \simeq B'_1 \\ e_{1*} : e_1 \sim_{\simeq^* A'_1 B'_1} e'_{1*} \end{array} \right). \\
& \quad \Pi^* \begin{bmatrix} a \\ a' \\ a_* \end{bmatrix} : A^* \Pi^* \begin{bmatrix} a_1 \\ a'_1 \\ a_1 \end{bmatrix} : A_1^* \Pi^* \begin{bmatrix} a_* \\ a'_* \\ a_* \end{bmatrix} : A_*^* \left(\begin{bmatrix} a' \\ a' \\ a_* \end{bmatrix} \right) \left(\begin{bmatrix} a_1 \\ a'_1 \\ a_1 \end{bmatrix} \right) \\
& \quad \Pi^* \begin{bmatrix} b \\ b' \\ b_* \end{bmatrix} : B^* \Pi^* \begin{bmatrix} b_1 \\ b'_1 \\ b_1 \end{bmatrix} : B_1^* \Pi^* \begin{bmatrix} b_* \\ b'_* \\ b_* \end{bmatrix} : B_*^* \left(\begin{bmatrix} b \\ b' \\ b_* \end{bmatrix} \right) \left(\begin{bmatrix} b_1 \\ b'_1 \\ b_1 \end{bmatrix} \right). \\
& \quad \simeq^* \left(e^* \left(\begin{bmatrix} a \\ a' \\ a_* \end{bmatrix} \right) \left(\begin{bmatrix} b \\ b' \\ b_* \end{bmatrix} \right) \right) \left(e_{1*} \left(\begin{bmatrix} a_1 \\ a'_1 \\ a_1 \end{bmatrix} \right) \left(\begin{bmatrix} b_1 \\ b'_1 \\ b_1 \end{bmatrix} \right) \right)
\end{aligned}$$

into

$$\begin{aligned}
& \simeq^* A_* B_* \sim_{((A \simeq B) \simeq (A_1 \simeq B_1))^*} \simeq^* A'_* B'_* \\
& = \simeq^* A_* B_* \sim_{\simeq^* (A \simeq B)^* \simeq (A_1 \simeq B_1)^*} \simeq^* A'_* B'_* \\
& = \prod \left(\begin{array}{c} e : A \simeq B \\ e' : A' \simeq B' \\ e^* : e \sim_{(A \simeq B)^*} e' \end{array} \right) \prod \left(\begin{array}{c} e_1 : A_1 \simeq B_1 \\ e'_1 : A'_1 \simeq B'_1 \\ e_1^* : e_1 \sim_{(A_1 \simeq B_1)^*} e'_1 \end{array} \right). \\
& \quad (e \sim_{\simeq^* A_* B_*} e_1) \simeq (e' \sim_{\simeq^* A'_* B'_*} e'_1)
\end{aligned}$$

The lambdas go into the pies quite easily, so we focus on

$$\begin{aligned}
(e \sim_{\simeq^* A_* B_*} e_1) \simeq (e' \sim_{\simeq^* A'_* B'_*} e'_1) & = \\
& \left(\begin{array}{c} \Pi a : A \Pi a_1 : A_1 \Pi a_* : a \sim_{A_*} a_1 \\ \Pi b : B \Pi b_1 : B_1 \Pi b_* : b \sim_{B_*} b_1. \quad a \sim_e b \simeq a_1 \sim_{e_1} b_1 \end{array} \right) \\
& \simeq \left(\begin{array}{c} \Pi a' : A' \Pi a'_1 : A'_1 \Pi a'_* : a' \sim_{A'_*} a'_1 \\ \Pi b' : B' \Pi b'_1 : B'_1 \Pi b'_* : b' \sim_{B'_*} b'_1. \quad a' \sim_{e'} b' \simeq a'_1 \sim_{e'_1} b'_1 \end{array} \right)
\end{aligned} \tag{11}$$

To inhabit this equivalence type, one needs to construct a sequence of equivalences which pairwise relate the domains of quantification in the sequence of Π -types on each side of the \simeq -sign.

Close inspection will reveal that the six Π^* -constructors appearing in the unfolding of $(\simeq^* A_* B_*)^*$ do provide such a sequence. For example, by induction hypothesis, we have

$$A_*^* : \prod \left(\begin{array}{c} x : A \\ x' : A' \\ x^* : x \sim_{A_*} x' \end{array} \right) \prod \left(\begin{array}{c} x_1 : A_1 \\ x'_1 : A'_1 \\ x_1^* : x_1 \sim_{A_1^*} x'_1 \end{array} \right). \quad x \sim_{A_*} x_1 \simeq x' \sim_{A'_*} x'_1$$

whence we get the equivalence

$$A_*^* \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \left(\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right) : a \sim_{A_*} a_1 \simeq a' \sim_{A'_*} a'_1$$

relating the domains of the third Π s in the sequence.

(Similarly, $B_*^* b b' b^* b_1 b'_1 b_1^* : b \sim_{B_*} b_1 \simeq b' \sim_{B'_*} b'_1$.)

All that remains is to check that

$$\begin{aligned}
& \simeq^* \left(e^* \left(\begin{array}{c} a \\ a' \\ a^* \end{array} \right) \left(\begin{array}{c} b \\ b' \\ b^* \end{array} \right) \right) \left(e_1^* \left(\begin{array}{c} a_1 \\ a'_1 \\ a_1^* \end{array} \right) \left(\begin{array}{c} b_1 \\ b'_1 \\ b_1^* \end{array} \right) \right) \\
& : (a \sim_e b \simeq a_1 \sim_{e_1} b_1) \simeq (a' \sim_{e'} b' \simeq a'_1 \sim_{e'_1} b'_1)
\end{aligned}$$

Indeed, this is attainable from

$$\begin{aligned}
e^* a a' a^* b b' b^* & : (a \sim_e b) \simeq (a' \sim_{e'} b') \\
e_1^* a_1 a'_1 a_1^* b_1 b'_1 b_1^* & : (a_1 \sim_{e_1} b_1) \simeq (a'_1 \sim_{e'_1} b'_1)
\end{aligned}$$

by feeding these terms into an application of the \simeq^* -rule.

This completes the proof of the theorem. \square

7. Extensional equality of ground types

We observe some consequences of the theorem.

1. Consider the type judgement

$$\vdash * : *$$

It is derivable by an axiom; by applying the theorem, we get

$$\vdash *^* : * \sim_{**} *$$

By the reduction rule for \simeq ,

$$* \sim_{**} * \longrightarrow * \simeq *$$

By conversion rule, the theorem is thus saying that

$$\vdash *^* : * \simeq *$$

which is indeed the case (axiom).

2. Consider a type judgement

$$\vdash A : *$$

Applying the theorem gives

$$\vdash A^* : A \sim_{**} A'$$

By conversion rule, we have $A^* : A \simeq A'$.

But A is closed term. So every variable of A is bound.

A' is obtained from A by apostrophizing every variable.

So A' is alpha equivalent to A .

And the type of A^* is alpha equivalent to $A \simeq A$.

DEFINITION. Let $\vdash A : *$ be a closed type. We define *extensional equality on A* to be

$$\sim_{A^*} : A \rightarrow A \rightarrow *$$

$$\boxed{a \simeq_A a' := a \sim_{A^*} a'}$$

A^* may be called *trivial type equality, identity equivalence* (on A), *reflexivity of A* .

3. Consider a type judgement

$$\vdash a : A$$

Applying the theorem gives

$$\vdash a^* : a \sim_{A^*} a'$$

Using the previous definition, we write this as

$$\vdash a^* : a \simeq_A a'$$

But a is closed term. So every variable of a is bound.

a' is obtained from a by apostrophizing every variable. So a' is α -equal to a .

And the type of a^* is α -equal to $a \simeq_A a$.

DEFINITION. Let $\vdash a : A$ be a closed term. We define the *reflexivity* of a to be

$$a^* \quad : \quad a \simeq_A a$$

$$\boxed{r(a) \quad := \quad a^*}$$

4. A particular case of the above is the judgment $\vdash A : *$

As before, we derive $r(A) = A^* : A \simeq_* A$. Then

$$a \simeq_A a' \quad = \quad a \sim_{r(A)} a'$$

For closed terms, the following rule is thus derived:

$$\boxed{\frac{\vdash a : A}{\vdash r(a) : a \sim_{r(A)} a}}$$

In particular, we derive $r(*) = *^* : * \simeq_* *$. Then

$$A \simeq_* B \quad = \quad A \sim_{r(*)} B \quad = \quad A \simeq B$$

$$\boxed{A \simeq_* B \quad = \quad A \simeq B}$$

5. Consider a type judgment

$$\vdash a : A$$

Using the theorem, we derive

$$\vdash r(a) : a \simeq_A a$$

Applying the theorem again gives

$$\begin{aligned} \vdash r(r(a)) & : \quad r(a) \simeq_{a \simeq_A a} r(a) \\ \vdash r(r(r(a))) & : \quad r(r(a)) \simeq_{r(a) \simeq_{a \simeq_A a} r(a)} r(r(a)) \\ & \vdots \end{aligned}$$

6. Suppose we have derivations

$$\begin{aligned} x : A \vdash b(x) : B(x) \\ \vdash a^* : a \simeq_A a' \end{aligned}$$

Applying the theorem, we get

$$\begin{aligned} x : A, x' : A, x^* : x \sim_{r(A)} x' \vdash B^*(x, x', x^*) : B(x) \simeq B(x') \\ x : A, x' : A, x^* : x \sim_{r(A)} x' \vdash b^*(x, x', x^*) : b(x) \sim_{B^*(x, x', x^*)} b(x') \end{aligned}$$

In particular, we obtain

$$\begin{aligned} \vdash B^*(a, a', a^*) : B(a) \simeq B(a') \\ \vdash b^*(a, a', a^*) : b(a) \sim_{B^*(a, a', a^*)} b(a') \end{aligned}$$

7. If we furthermore have

$$x : A, y : B(x) \vdash c(x, y) : C(x, y)$$

then we also obtain

$$\begin{aligned} & \vdash C^*(a, a', a^*, b(a), b(a'), b^*(a, a', a^*)) : C(a, b(a)) \simeq C(a', b(a')) \\ & \vdash c^*(\vec{a}, \overrightarrow{b(a)}) : c(a, b(a)) \sim_{C^*(\vec{a}, \overrightarrow{b(a)})} c(a', b(a')) \end{aligned}$$

DEFINITION. Let $\Gamma = (x_1 : A_1, \dots, x_n : A_n(x_1, \dots, x_{n-1}))$ be a context. A *path in Γ* is a sequence of terms

$$(\vec{a}, \vec{a}', \vec{a}^*) = (a_1, a'_1, a_1^*, \dots, a_n, a'_n, a_n^*)$$

such that, for each $i \in \{0, \dots, n-1\}$, the following holds:

$$\begin{aligned} & \vdash a_i : A_i(a_1, \dots, a_{i-1}) \\ & \vdash a'_i : A_i(a'_1, \dots, a'_{i-1}) \\ & \vdash a_i^* : a_i \sim_{A_i^*(a_1, a'_1, a_1^*, \dots, a_{i-1}, a'_{i-1}, a_{i-1}^*)} a'_i \end{aligned}$$

When $(\vec{a}, \vec{a}', \vec{a}^*)$ is a path in Γ , we write

$$\vec{a}^* \quad : \quad \vec{a} \simeq_{\Gamma} \vec{a}'$$

COROLLARY. The theorem of Section 6 has the following consequences.

- Every ground type possesses the structure of a globular set with degeneracies.
- If $\Gamma \vdash B(x_1, \dots, x_n) : *$, then every path $(\vec{a}, \vec{a}', \vec{a}^*)$ in Γ induces a type equality

$$\vdash B(\vec{a}^*) : B(\vec{a}) \simeq B(\vec{a}')$$

- If $\Gamma \vdash b(\vec{x}) : B(\vec{x})$, then every path $(\vec{a}, \vec{a}', \vec{a}^*)$ in Γ induces an equality over $B^*(\vec{a}, \vec{a}', \vec{a}^*)$:

$$\vdash b(\vec{a}^*) : b(\vec{a}) \sim_{B(\vec{a}^*)} b(\vec{a}')$$

8. Stratification and semantics

8.1. Stratification of $\lambda \simeq$

DEFINITION. The system $\lambda \simeq_n$ is obtained from $\lambda \simeq$ by executing the following recipe:

1. The symbol $*$ is replaced by an infinite collection of constants

$$\{\ast_n \mid n \in \omega\}$$

2. The typing rule $\overline{\vdash \ast : \ast}$ is replaced by the rule scheme (one rule for each number n):

$$\overline{\vdash \ast_n : \ast_{n+1}}$$

3. A new rule is introduced:

$$\frac{\Gamma \vdash A : *_{n+1}}{\Gamma \vdash A : *_{n+1}}$$

4. In all other rules, the $*$ symbol is replaced by $*_{n+1}$.

REMARK. When we describe the intended model, it will turn out that the above definition is not quite correct: the elimination rule for \simeq does not have the right universe indexing. We shall address this issue when it arises in the course of our construction.

8.2. The strict model

First, we describe a particularly simple model in which $A \simeq B$ is interpreted by strict, set-theoretic equality. This model even validates the rule derivable in the Martin-Löf (1984) system with propositional reflection:

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash e : A \simeq B}{\Gamma \vdash a : B}$$

Let $\kappa_0 < \kappa_1 < \dots$ be a sequence of strongly inaccessible cardinals.

1. Each universe $*_n$ is interpreted by $\mathbf{Set}_n = V_{\kappa_n}$, the cumulative hierarchy up to stage κ_n :

$$\llbracket *_n \rrbracket := \mathbf{Set}_n$$

2. The Π - and Σ -types are interpreted, respectively, by cartesian product and disjoint union of families of sets:

$$\begin{aligned} \llbracket \Pi x:A. B(x) \rrbracket &:= \prod_{a \in \llbracket A \rrbracket} \llbracket B \rrbracket_{x:=a} = \{ f : \llbracket A \rrbracket \rightarrow \bigcup_{a \in \llbracket A \rrbracket} \llbracket B \rrbracket_{x:=a} \mid \forall a. f a \in \llbracket B \rrbracket_{x:=a} \} \\ \llbracket \Sigma x:A. B(x) \rrbracket &:= \bigsqcup_{a \in \llbracket A \rrbracket} \llbracket B \rrbracket_{x:=a} = \{ (a, b) \mid a \in \llbracket A \rrbracket, b \in \llbracket B \rrbracket_{x:=a} \} \end{aligned}$$

3. The \simeq -type is interpreted by equality:

$$\llbracket A \simeq B \rrbracket := \begin{cases} \{\emptyset\} & \llbracket A \rrbracket = \llbracket B \rrbracket \\ \emptyset & \llbracket A \rrbracket \neq \llbracket B \rrbracket \end{cases}$$

Since, for κ strongly inaccessible, V_κ is closed under cartesian products and disjoint union, the above definition manifestly validates the four formation rules of λ^*_{κ} , as well as the subsumption rule.

The interpretation of term formers related to the Π - and Σ -types is completely standard:

$$\begin{aligned} \llbracket \lambda x:A. t \rrbracket_\rho &= (a \mapsto \llbracket t \rrbracket_{\rho, x:=a}) && \in \prod_{a \in \llbracket A \rrbracket_\rho} \llbracket B \rrbracket_{\rho, a} \\ \llbracket f a \rrbracket_\rho &= \llbracket f \rrbracket_\rho(\llbracket a \rrbracket_\rho) \\ \llbracket (a, b) \rrbracket_\rho &= (\llbracket a \rrbracket_\rho, \llbracket b \rrbracket_\rho) && \in \bigsqcup_{a \in \llbracket A \rrbracket_\rho} \llbracket B \rrbracket_{\rho, a} \\ \llbracket \pi_i t \rrbracket_\rho &= p_i \quad \text{where } (p_1, p_2) = \llbracket t \rrbracket_\rho \end{aligned}$$

The interpretation of $\Pi^*, \Sigma^*, \simeq^*, *$ is self-evident. When $e : A \simeq B$, put

$$\llbracket a \sim_e b \rrbracket \quad := \quad \begin{cases} \{\emptyset\} & \llbracket a \rrbracket = \llbracket b \rrbracket \\ \emptyset & \text{otherwise} \end{cases}$$

The interpretation of contexts Γ is the set of all tuples (a_1, \dots, a_n) such that

$$a_{i+1} \in \llbracket A_{i+1} \rrbracket_{a_1, \dots, a_i} \quad (0 \leq i < n)$$

It is straightforward to verify that the interpretation preserves substitution, conversion, and typing rules. So we have

THEOREM. (Soundness) Let $\llbracket \Gamma \rrbracket, \llbracket A \rrbracket_{\rho \in \llbracket \Gamma \rrbracket}, \llbracket a \rrbracket_{\rho \in \llbracket \Gamma \rrbracket}$ be as defined above. Then

$$\Gamma \vdash M : A \quad \Longrightarrow \quad \llbracket M \rrbracket : \prod_{\tilde{a} \in \llbracket \Gamma \rrbracket} \llbracket A \rrbracket_{\tilde{a}}$$

The above model is *proof-irrelevant*, since proofs of equality have no computational content. It is in keeping with our goal of generality, that extensional equality should admit such an interpretation.

However, the \simeq -type also contains all the necessary machinery for transporting computational information over proofs of equality. We shall now describe a model which makes use of this feature.

Due to lack of space, we do not go into details, but give a general outline.

8.3. The proof-relevant model

8.3.1. Isomorphism as equality

In this model, type equality is interpreted as isomorphism of sets:

$$\llbracket A \simeq B \rrbracket \quad := \quad \llbracket A \rrbracket \simeq \llbracket B \rrbracket$$

Let $\{B_x \mid x \in A\}, \{B'_y \mid y \in A'\}$ be families of sets. Given an isomorphism $i : A \xrightarrow{\simeq} A'$, and a family $\{j_{a,a'} : B_a \xrightarrow{\simeq} B_{a'} \mid i(a) = a'\}$, we obtain isomorphisms

$$\begin{aligned} \Pi_{i(x)=y}^*(j_{x,y}) & : \prod_{x \in A} B_x & \xrightarrow{\simeq} & \prod_{y \in A'} B'_y \\ \sqcup_{i(x)=y}^*(j_{x,y}) & : \bigsqcup_{x \in A} B_x & \xrightarrow{\simeq} & \bigsqcup_{y \in A'} B'_y \end{aligned}$$

Given isomorphisms $i : A \xrightarrow{\simeq} A', j : B \xrightarrow{\simeq} B'$, we get an isomorphism (conjugation):

$$\begin{aligned} \simeq^*(i, j) & : (A \simeq B) & \longrightarrow & (A' \simeq B') \\ \simeq^*(i, j) & : \xi & \longmapsto & j \circ \xi \circ i^{-1} \end{aligned}$$

Every isomorphism $i : A \simeq B$ induces a binary relation $\tilde{i} \subseteq A \times B$:

$$a \tilde{i} b \quad \Longleftrightarrow \quad i(a) = b$$

Every set A has the identity isomorphism:

$$I_A : A \simeq A$$

In particular, there exist canonical isomorphisms

$$I_{\mathbf{Set}_n} : V_{\kappa_n} \simeq V_{\kappa_n}$$

This fixes the interpretation of everything related to type equality. Keeping the interpretation of other types the same, we now try to validate the reduction rules.

8.3.2. A bug?

Thankfully, all of the reduction rules are perfectly valid in our model.

Except one.

$$A \sim_{**} B \longrightarrow A \simeq B \tag{12}$$

The right side is interpreted by the set of isomorphisms $\llbracket A \rrbracket \simeq \llbracket B \rrbracket$.

The left side is interpreted by the relation $I_{\mathbf{Set}_n} \subseteq \mathbf{Set}_n \times \mathbf{Set}_n$:

$$\llbracket A \sim_{**} B \rrbracket = \begin{cases} \{\emptyset\} & \llbracket A \rrbracket = \llbracket B \rrbracket \\ \emptyset & \llbracket A \rrbracket \neq \llbracket B \rrbracket \end{cases}$$

The rule (12) is thus saying that there is at most one isomorphism between any pair of sets, a claim which many members of \mathbf{Set}_n will find offensive.

How are we to reconcile this reduction rule with our model?

8.3.3. The truth-table universe

In the first instance, we notice that the offending claim *does* make sense for sets that are either empty or singletons, i.e., *propositions*.

Indeed, the universe $\{0, 1\}$ of classical propositions is closed under isomorphism, in the sense that there is indeed at most one isomorphism between any two propositions — its existence being equivalent to the existence of a pair of maps between them.

Furthermore, all other type constructors can be given the standard “truth-table semantics” in this universe, validating their introduction and elimination rules.

This motivates us to let propositions actualize the interpretation of \ast_0 , the lowest universe in λ_{\simeq_n} . Constructions carried out in this universe fall in the scope of the *propositions as types* embedding (Howard (1980)).

8.3.4. Homotopy hierarchy

We are thus led to reconsider our interpretation

$$\llbracket \ast_n \rrbracket = \mathbf{Set}_n, \quad \llbracket \ast_n^\ast \rrbracket = \mathbf{Set}_n^\ast = I_{\mathbf{Set}_n} : \mathbf{Set}_n \xrightarrow{\simeq} \mathbf{Set}_n$$

In order for (12) to remain valid while preserving “type equality is isomorphism” idea, we *must* observe

$$\llbracket A \rrbracket \widetilde{\mathbf{Set}^\ast} \llbracket B \rrbracket = \llbracket A \rrbracket \simeq \llbracket B \rrbracket$$

This means that the equality on **Set** — which we defined as the “relation” induced by identity equality of **Set** with itself — must actually be a *set family* (giving, for any two sets, the set of isomorphisms between them), rather than a simple relation.

In contrast, for A, B elements of **Set**, the relation $\tilde{e} \subseteq A \times B$ induced by an isomorphism $e : A \xrightarrow{\sim} B$, is always two-valued: it’s a proposition. In particular, the equality relation induced by the identity isomorphism on A is a proposition.

For propositions, an “isomorphism” is just a pair of maps, and the relation associated to this pair is the total (“1-valued”) relation between the two propositions.

Going higher, we find that, for groupoids G_1, G_2 , the collection of groupoid equivalences between G_1 and G_2 forms again a groupoid⁶, and every groupoid equivalence $E : G_1 \Rightarrow G_2$ induces a **Set**-valued predicate on $G_1 \times G_2$:

$$\tilde{E} := (A \in \text{Ob}(G_1)) \mapsto (B \in \text{Ob}(G_2)) \mapsto \text{Hom}_{G_2}(E(A), B)$$

We observe the following pattern:

The relation $\sim e : A \rightarrow B \rightarrow *$ induced by a type equality $e : A \simeq B$ between types in the universe $*_n$, is valued in the universe $*_{n-1}$.

This pattern leads us to revoke the interpretation of universes in the cumulative hierarchy of set theory in favor of the (still cumulative) hierarchy of homotopy n -types:

$$\begin{aligned} \llbracket *_{\mathbf{0}} \leq *_{\mathbf{1}} \leq *_{\mathbf{2}} \leq \dots \leq *_{\mathbf{n}} \leq \dots \rrbracket &:= \text{Prop} \subseteq \text{Set} \subseteq \text{Grpd} \subseteq \dots \subseteq (n-2)\text{-Grpd} \subseteq \dots \\ &= \text{Prop} \in \text{Set} \in \text{Grpd} \in \dots \in (n-2)\text{-Grpd} \in \dots \end{aligned}$$

8.3.5. Fixing the bug

The pattern announced above forces us to reconsider the \simeq -elimination rule in the stratified system. It shall now be read as follows.

$$\frac{A : *_n \quad e : A \simeq B \quad B : *_n}{\sim e : A \rightarrow B \rightarrow *_{n-1}}$$

Postponing for the moment the question of what we are to make of the conclusion in the case when $n = 0$, we point out that this change resolves the problem in (12), allowing us to complete the model. Thus, for $A, B : *_n$, we have

$$\begin{aligned} \llbracket A \simeq B \rrbracket &= n\text{-equivalence between } n\text{-types,} \\ \llbracket A \sim_{*_n} B \rrbracket &= n\text{-relation induced by the identity equivalence of the } (n+1)\text{-type } *_n. \end{aligned}$$

These collections may be naturally identified. In particular:

- Two propositions are equivalent if their truth-table semantics yield isomorphic sets;
- Two sets A, B are isomorphic if the identity groupoid equivalence $\text{Id}_{\text{Set}} : \text{Set} \xrightarrow{\sim} \text{Set}$ relates them as objects: $(A \simeq B) = \text{Set}(\text{Id}_{\text{Set}}(A), B)$.
- etc.

⁶ Given $E, E' : G_1 \xrightarrow{\sim} G_2$, the isomorphisms between E, E' may be given equivalently either as

$$\prod_{A \in G_1} \prod_{B \in G_2} G_2(E(A), B) \simeq G_2(E'(A), B) \quad \text{or} \quad \prod_{A \in G_1} G_2(E(A), E'(A))$$

These collections, being products of sets, are again sets.

We remark that the formation rule of the equality type needs no amendments:

$$\frac{A : *_{\mathbf{1}} \quad B : *_{\mathbf{1}}}{A \simeq B : *_{\mathbf{1}}}$$

This rule already gives the intended meaning:

- Propositions are closed under logical equivalence;
- Sets are closed under isomorphism;
- Groupoids are closed under equivalence of groupoids;
- etc.

Although we have discussed only the first few levels of the homotopy hierarchy, it is clear that the given pattern has a clear inductive structure, and extends to all finite n -types.

We may also consider adding a “limit universe”

$$*_{\mathbf{0}} \leq *_{\mathbf{1}} \leq *_{\mathbf{2}} \leq *_{\mathbf{3}} \leq \dots \leq *_{\omega}$$

for which the $\sim(\cdot)$ -operator would stay valued in $*_{\omega}$. The natural interpretation of $*_{\omega}$ would be by a model of weak ω -groupoids.

But in order for such a universe to be of any interest, our language should already provide the computational interpretation of the higher groupoid laws (the Kan filling conditions). This of course is a major topic for future work.

To complete our model, we discuss the rule of \simeq -elimination for the case when $n = 0$.

The symbol $*_{-1}$ is treated as notation for $\mathbf{1}$, the unit type.

We add new formation, introduction, elimination, and computation rules for this type.

$$\frac{\overline{\Gamma \vdash \mathbf{1} : *_{\mathbf{0}}} \quad \overline{\Gamma \vdash t : \mathbf{1}}}{\frac{\Gamma, x : \mathbf{1} \vdash B : * \quad \Gamma \vdash b : B[t/x]}{\Gamma \vdash \text{Const}_B(b) : \prod x : \mathbf{1}. B}}$$

$$\text{Const}_B(b)t \longrightarrow b$$

We add the rules for interaction between $\mathbf{1}$ and \simeq .

$$\frac{\overline{\Gamma \vdash \mathbf{1}^* : \mathbf{1} \simeq \mathbf{1}}}{x \sim_{\mathbf{1}^*} y \longrightarrow \mathbf{1}}$$

Finally, we extend the $(\cdot)^*$ -operation to these new terms:⁷

$$\begin{aligned} (\mathbf{1})^* &= \mathbf{1}^* \\ (t)^* &= t \quad (: t \sim_{\mathbf{1}^*} t) \\ (\mathbf{1}^*)^* &= \lambda \left(\begin{array}{c} x : \mathbf{1} \\ x' : \mathbf{1} \\ x^* : x \simeq_{\mathbf{1}} x' \end{array} \right) \lambda \left(\begin{array}{c} y : \mathbf{1} \\ y' : \mathbf{1} \\ y^* : y \simeq_{\mathbf{1}} y' \end{array} \right). \mathbf{1}^* \end{aligned}$$

This completes our model construction.

⁷ We do not include the lifting of the **Const** eliminator; in future versions of our system **Const** is expected to be derivable from the *transport operator*:

$$\frac{\frac{\frac{\Gamma, x \in \mathbf{1} \vdash B(x) : * \quad \Gamma \vdash u : \mathbf{1}}{\Gamma \vdash B^*(t, u, t) : B(t) \simeq B(u)}}{\Gamma \vdash B^*(t, u, t)^+ : B(t) \rightarrow B(u)} \quad \Gamma \vdash b : B(t)}{\Gamma \vdash B^*(t, u, t)^+ b : B(u)}$$

9. Conclusion

In this paper, we have enunciated Tait’s suggestion for the type-theoretic meaning of the notion of extensional equality.

We have shown how the external definition of extensional equality may be reflected into the syntax. Our construction yields an internal definition of extensional equality for closed types.

We have not yet witnessed all of the desired properties of this equality. Future work includes generalization to open terms and computational treatment of Kan filling conditions (to be defined via transport maps over type equality).

In the model outlined in the last section, we see the syntactic approach to extensional equality starting to come together with the threads of ideas motivated by homotopy theory.

An original feature of this interpretation is that the logical relation defined by induction on type structure is reflected in the *lower* universe than the types being related by it.

References

- Altenkirch, Thorsten (1999). Extensional equality in intensional type theory, *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, IEEE Computer Society, pp. 412–420.
- Altenkirch, Thorsten, Conor McBride and Wouter Swierstra (2007). Observational equality, now!, *in*: Aaron Stump and Hongwei Xi (eds.), *PLPV*, ACM, pp. 57–68.
- Bernardy, Jean-Philippe and Guilhem Moulin (2012). A computational interpretation of parametricity, *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, LICS ’12, IEEE Computer Society, Washington, DC, USA, pp. 135–144.
- Bezem, Marc, Thierry Coquand and Simon Huber (2014). A Model of Type Theory in Cubical Sets, *in*: Ralph Matthes and Aleksy Schubert (eds.), *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, Leibniz International Proceedings in Informatics (LIPIcs) 26, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 107–128.
- Coquand, Thierry (2011). Equality and dependent type theory.
<http://www.cse.chalmers.se/~coquand/equality.pdf>.
- Gandy, Robin O. (1956). On the axiom of extensionality–part I, *J. Symb. Log.* **21**(1), pp. 36–48.
- Gonthier, Georges, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O’Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi and Laurent Théry (2013). A machine-checked proof of the odd order theorem, *in*: Sandrine Blazy, Christine Paulin-Mohring and David Pichardie (eds.), *ITP*, Lecture Notes in Computer Science 7998, Springer, pp. 163–179.
- Hofmann, Martin and Thomas Streicher (1996). The groupoid interpretation of type theory, *In Venice Festschrift*, Oxford University Press, pp. 83–111.

- Howard, William A. (1980). The formulas-as-types notion of construction, *in*: J. P. Seldin and J. R. Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press, pp. 479–490.
- Licata, Daniel R. and Robert Harper (2012). Canonicity for 2-dimensional type theory, *in*: John Field and Michael Hicks (eds.), *POPL*, ACM, pp. 337–348.
- Martin-Löf, Per (1984). *Intuitionistic Type Theory*, Bibliopolis, Naples.
- Martin-Löf, Per (2013). Invariance under isomorphism and definability, The 2013 Ernest Nagel Lectures in Philosophy and Science.
- Polonsky, Andrew (2014). Extensionality of lambda-*, <http://arxiv.org/abs/1401.1139>.
- Sozeau, Matthieu and Nicolas Tabareau (2014). Towards A Mechanized Model of Type Theory Based On Groupoids.
- Tait, William W. (1995). Extensional equality in the classical theory of types, *in*: Werner Depauli-Schimanovich, Eckehart Khler and Friedrich Stadler (eds.), *The Foundational Debate*, Vienna Circle Institute Yearbook [1995] 3, Springer Netherlands, pp. 219–234.
- The Univalent Foundations Program, I.A.S. (n.d.). *Homotopy Type Theory: Univalent Foundations of Mathematics*, Univalent Foundations.
- Voevodsky, Vladimir (2006). A very short note on the homotopy λ -calculus, http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/Hlambda_short_current.pdf.